# Hangman

## Step 1 – Selecting A Word At Random

 Dim wordList() As String = {"sandwich", "apple", "steak", "chips", "lentils"}

 Dim randomNumberGenerator As New Random

 Dim selectedWord As String

 Dim wordNum As Integer

 wordNum = randomNumberGenerator.Next(wordList.Length)

 selectedWord = wordList(wordNum)

 Console.WriteLine(selectedWord)

 Console.ReadLine()

1. Copy this code and run it. It does the job that you need for this stage. The first statement declares an array. An array is a single variable name that stores multiple items. Change the array to choose words that you want in the game. Make sure that there are around a dozen words for the game.
2. Run the program a few times and make sure that you get different words over time.
3. You could make it easier to test if you added a FOR loop to the code to choose words, say 20 times. Document your experiment with this and then remove the repetition statements.

## Step 2 – Create Dash Pattern

The dash pattern is used in hangman to show the player how many letters there are in the word they have to guess. Here is some pseudocode that outputs a dash for every letter in the randomly selected word.

Declare wordLength as integer

Declare dashPattern as String ← “”

wordLength ← selectedWord.Length

For i ← 0 to wordLength -1

 dashPattern ← dashPattern + “-“

End For

Output dashPattern

We use pseudocode to describe algorithms and programs so that our descriptions can be used with any programming language. When you write a program from pseudocode, you have to make sure that you are writing the correct format for the statement in the programming language you are using. ←means the same as an equals sign in Visual Basic. It is written this way in pseudocode when we assign variables. This is so that we can distinguish between assignment and comparison statements.

1. The variable declarations should go with the other ones from the first step. The other lines come after the code that selects a random word, before that word is displayed. Enter the correct code.
2. Test the program and make sure that you are getting the same number of dashes as the number of letters in the word.

## Step 3 – Make The Game

Declare guessed as String ← “”

Declare lives as Integer ← 6

Declare win as Boolean ← false

Declare thisGuess as String

Declare newDash as String

Do

 Output dashPattern

 Output “Guessed: “ + guessed

 Output “Lives: “ + lives

 Output “Guess a letter from the word”

 Input thisGuess

 If selectedWord.Contains(thisGuess) then

 newDash = “”

 For i ← 0 to wordLength -1

 If selectedWord(i) = thisGuess Then

 newDash ← newDash + thisGuess

 Else

 newDash ← newDash + dashPattern(i)

 End If

End For

 dashPattern ← newDash

 Else

 If guessed.Contains(thisGuess) **= false** then

 guessed ← guessed + thisGuess

 lives ← lives – 1

 End If

 End If

 If dashPattern = selectedWord Then

win ← true

Output “Winner”

 End If

 If lives = 0 Then

 Output “Loser”

 End If

Loop Until lives = 0 Or win =true

1. Before you enter any of this code, you need to remove the two lines that displayed the word and the dash pattern. Add the variable declarations to the section you have at the top of the program where you have the other declarations.
2. Remember to press the enter key at the end of each line and to look up at the screen. When you enter DO, IF, FOR statements, Visual Basic will add the ending part of the structure for you. This means that, after you press enter, you should not see the blue underlining that indicates an error in the code.
3. Test, test, test.

## Step 4 – Improve

1. Get a good size word list. Check that your words are all lower case and add them to your program.
2. Add code so that the user can play more than one game without having to rerun the program.
3. Add code so that, after each game, the user gets to see the number of wins and losses.

## Step 5 – Challenge 1 – 2 Player

Copy the code that you have and paste it into a new solution, Hangman 2. This version of the game will be for two players.

Player 1 enters a word. Player 2 plays the guessing game. They each have a score. If Player 2 guesses the word within the required number of goes, they get a point, otherwise Player 1 gets a point.

Start by changing the way a word is selected so that it is a word entered by player 1. Remove the lines of code that are no longer needed in the program.

Test that the new word is being used correctly.

Change the program so that the Player names and scores are displayed and that it is clear who should be entering things at the console.

## Step 6 – Challenge 2 – Better 2 Player

Add a variable in your program to store who is guessing, Player 1 or Player 2. Use this variable to make the game alternate between the players for the setter and guesser roles.

After each word is guessed or the player runs out of guesses, show an updated score for each player.

## Step 6 – Challenge 3 – Rounds

At the start of the program, the user should enter a number of rounds that they want to play. A round means that Player 1 enters a word that Player 2 tries to guess. After that, Player 2 enters a word for Player 1 to guess.

When the correct number of complete rounds have been played, the game should end and the final scores be displayed. If there is an outright winner in the scores, this should be displayed. If the match results in a draw, this should be displayed too.

## Step 7 – Challenge 4 – Hangman 2000

Your choice, your challenge.

You could find a different way of scoring each round of the game. If they both guess correctly or both fail to guess, the round is a draw. If one guesses, the other fails, then the guesser wins the round.

You could make your game a Hangman Penalty Shoot-Out. Play 5 rounds and then declare a winner. If there is no clear winner after 5 rounds, the game goes to sudden death. Do this in two stages. Start by working out how to implement the sudden death feature to stop the game. The game stops if the round wasn’t drawn. When you have this, change it so that the game stops if 5 rounds have been played and the player scores are not equal.